Some Additional Remarks on ADTs and Modules in ANSI C

Suppose you wish to implement an ADT in C. The particular ADT is unimportant, so let's just call it a "Blah". You should create the following files at minimum: Blah.c, Blah.h, BlahTest.c. Blah.c represents the 'inside' of the black box, Blah.h represents the interface, and BlahTest.c is a kind of dummy client module used to wring the bugs out of the Blah ADT.

File Blah.h will contain prototypes for all exported functions which represent ADT operations. It will also contain the line.

```
typedef struct Blah*  BlahRef;
```

This defines `BlahRef` to be a pointer to some struct called `Blah`. Blah.c will #include Blah.h, and will contain definitions of all exported functions, and perhaps definitions of some private functions and structs as well. Blah.c will also contain the following typedef statement.

```
typedef struct Blah{
   /* code which defines fields for the Blah ADT */
} Blah;
```

A client module can then #include Blah.h giving it the ability to declare variables of type BlahRef, as well as functions which take BlahRef parameters. However, the client cannot dereference a BlahRef pointer, since the object it points to is not defined in Blah.h. The ADT operations take BlahRef arguments, so the client does not need to (and is in fact unable to) directly access the struct which these references point to. Therefore the client can interact with a Blah object only through the exported ADT operations. This is how information hiding is accomplished in C.

Blah.c also contains a constructor:

```
BlahRef newBlah(...){
   BlahRef B;
   /* code which initializes B */
   return(B);
}
```
and a destructor
```
void freeBlah(BlahRef* pB){
   if(pB!=NULL && *pB!=NULL){
      /* free all heap memory associated with *pB */
      free(*pB);
      *pB = NULL;
   }
}
```

Notice that the destructor is defined in a strange way. It's argument is not a BlahRef, but a pointer to a BlahRef (i.e. a pointer to a pointer.) Therefore the destructor is called by passing the address of a BlahRef. Each Blah object is explicitly created and destroyed as follows.

```
BlahRef B = newBlah(...);
/* do something with B */
freeBlah(&B);
```

Function `freeBlah()` must be defined in this way (i.e. taking a pointer to BlahRef rather than a simple BlahRef) since it is the one ADT operation that changes the BlahRef itself (in addition to the Blah it points to) by setting it to NULL.

As in java, all ADT operations must check their own preconditions and exit with a useful error message when one of them is violated. What should the message say? It should state the module in which the error occurred (i.e. Blah), the operation in which it occurred, and exactly which precondition was violated. The purpose of this message is to provide diagnostic assistance to whoever is programming a client of the Blah ADT. In this class that person is of course you the student, but in a the real world that may well be another programmer, so you must make the error message as helpful as possible.

In C however, there is one more item to check. Each ADT operation should check that the reference which is it's main argument is not NULL. This check should come before the checks of preconditions, since any attempt to dereference a NULL pointer will result in a segmentation fault.

```
void some_op(BlahRef B){
    if(B==NULL){
        printf("Blah Error: calling some_op on NULL BlahRef");
        exit(1);
    }
    /* check preconditions and do stuff */
}
```

This was not necessary in java was because calling an instance method on a null reference variable causes a NullPointerException to be thrown, providing some error tracking to the programmer.

Finally a word about our naming conventions in C. In some other programming classes you may have used names like 'BlahHndl' or 'BlahPtr' instead of 'BlahRef'. Obviously one name is not intrinsically better than another, but for the sake of consistency, you are required to adhere to the naming conventions outlined in this document and the previous ADT handout. In particular the file names Blah.c, Blah.h, BlahTest.c; the function names `newBlah()`, `freeBlah()`, `printBlah()`; and the type names Blah, and BlahRef are not open to modification. This is not unlike the situation programmers face in industry. Each company has some set of notational conventions and procedures which everyone follows so that everybody can easily read and maintain each other's work.

2